MP2I
2021/2022

Informatique
Devoir surveillé 1
2 octobre 2021

CIV

Durée totale 4h.
Tout document interdit. Tout dispositif électronique interdit.
On vous demande d'écrire lisiblement et de bien présenter. Les programmes en particulier doivent être bien indentés et lisibles.
Bon courage et *Don't panic !* La barème et les indications de temps sont purement indicatives.

1 Reprise des TPs - 10pts

On demande ici de redonner les programmes correspondant plus ou moins directement à des questions posées dans les TP1 à 4.

1.1 OCaml - 5pts

Les programmes à donner en réponse ici doivent être écrits en `OCaml`.

! on utilise ici les types par défaut de `OCaml`. C'est particulièrement important pour les listes.

Q01 **OCaml - 5min - 1/2pt** Écrire une fonction `pow2 : int -> int` telle que `pow2 n` renvoie 2^n où $n \geq 0$ (c'est acquis que n ne sera jamais négatif donc ne vous en préoccupez pas dans les questions.).

Q02 **OCaml - 5min - 1/2pt** Écrire une fonction `suplog2 : int -> int` telle que `suplog2 n` renvoie le plus petit $d \in \mathbb{N}$ tel que $n < 2^d$. On supposera $n \geq 0$.

Q03 **OCaml - 5min - 1/2pt** Écrire une fonction `comb : int -> int -> int` telle que `comb n k` calcule $\binom{n}{k}$ avec la relation du triangle de Pascal :

$$\forall n \in \mathbb{N}^*, \forall k \in \{1, \dots, n\}, \quad \binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

On rappelle que $\binom{n}{0} = \binom{n}{n} = 1$ et $\binom{n}{k} = 0$ par convention quand $k \notin \{0, \dots, n\}$.

Q04 **OCaml - 5min - 1/2pt** Écrire une fonction `trib : int -> int` telle que `trib n` calcule T_n où $T_0 = T_1 = 0$, $T_2 = 1$ et

$$\forall n \in \mathbb{N}, T_{n+3} = T_{n+2} - T_n T_{n+1}$$

Q05 **10min - 3/4pt** On considère C_n le nombre de noeuds dans l'arbre d'appels récursifs de la fonction `trib`. Déterminer une relation de récurrence satisfaite par la suite $(C_n)_{n\in\mathbb{N}}$.

Q06 **OCaml - 10min - 3/4pt** Écrire une fonction `applique : 'a list -> ('a -> 'b) -> 'b list` telle que l'appel à `applique l f` renvoie la liste obtenue en appliquant à chaque élément de `l` la fonction `f`.
Exemple :

```
applique [1;2;3] (fun x -> 4 + x)
(* renvoie *)
[5;6;7]
```

Q07 **OCaml - 10min - 3/4pt** Écrire une fonction `nieme : 'a list -> int -> 'a option` telle que l'appel à `nieme l n` renvoie `Some x` où `x` est le n-ième élément de `l` ou `None` s'il n'y en a pas.

Q08 **OCaml - 10min - 3/4pt** Écrire une fonction `pourtout : 'a list -> ('a -> bool) -> bool` telle que l'appel à `pourtout l f` renvoie `true` *si et seulement si* tous les éléments `x` de `l` sont tels que `f x` soit égale à `true`.

1.2 C - 5pts

Les programmes à donner en réponse ici doivent être écrits en `C`.

Q09 **C - 5min - 1/2pt** Écrire une fonction de signature `int suplog2(int n)` qui renvoie le plus entier $d \geq 0$ tel que $0 \leq n < 2^d$.

Q10 **C - 5min - 1/2pt** Écrire une fonction de signature `int factorielle(int n)` qui renvoie $n!$.



on ne fera pas de fonction récursive.

Q11 **C - 5min - 1/2pt** Écrire une fonction de signature `int somme_cubes(int t[], int nb_elts)` qui calcule la somme des cubes des éléments du tableau donné en arguments.

Q12 **C - 2min - 1/4pt** Écrire une fonction de signature `void swap(int t[], int i, int j)` qui échange les valeurs aux indices `i` et `j` dans le tableau `t`.

Q13 **C - 10min - 1pt** Écrire une fonction de signature `int indice_minimum(int t[], int i, int j)` qui renvoie l'indice du plus petit élément situé entre les indices i et j (inclus).
On supposera que la fonctions sera toujours appelée avec $0 \leq i < j < n$ où n est le nombre d'éléments du tableau.

Q14 **C - 10min - 1pt** En déduire une fonction de signature `void tri(int t[], int nb_elts)` qui trie le tableau donné en argument à l'aide des deux fonctions précédentes.

Q15 **2min - 1/4pt** Comment s'appelle ce tri ?

Q16 **C - 10min - 1pt** Écrire une fonction de signature `int recherche_motif(char s[], char m[])` qui renvoie l'indice de la première occurrence de la chaine `m` dans la chaîne `s` si elle s'y trouve, ou -1 dans le cas contraire.

2 Autour du crible d'Eratosthène - 2pts

On cherche ici à déterminer tous les nombres premiers $\leq N$ où N est une limite qu'on se fixe à l'avance.
L'idée du crible d'Eratosthène consiste à aligner tous les nombres de 2 à N, puis d'effectuer le processus suivant :

1. Déterminer le plus petit entier n non barré et non encore étudié.
2. Barrer tous les multiples de n qui sont $> n$.
3. S'il reste des nombres non barrés $> n$ retourner à l'étape 1.
4. Les nombres non barrés sont premiers.

Par exemple pour $N = 9$:

a) initialisation : 2 3 4 5 6 7 8 9

b) étape 1 – $n = 2$ on barre donc $4, 6, 8$: 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9

c) étape 2 – $n = 3$ on barre donc $6, 9$ (mais 6 est déjà barré) : 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~

d) étape 3 – $n = 5$ mais il n'y a pas de multiples : 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~

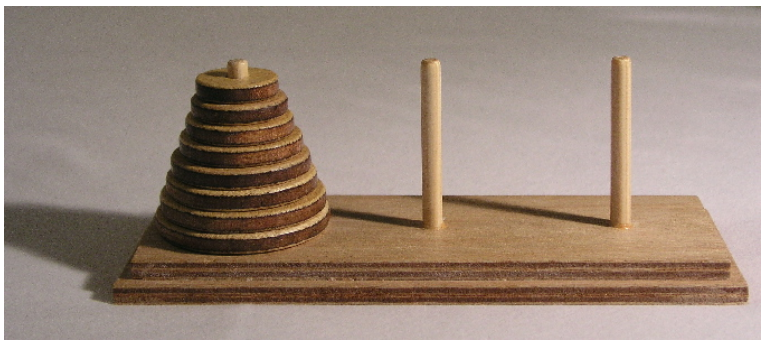
e) étape 4 – $n = 7$ mais il n'y a pas de multiples : 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~

f) étape 5 – il n'y a plus de nombres non barrés, on a donc trouvé les premiers ≤ 9 : 2 3 5 7

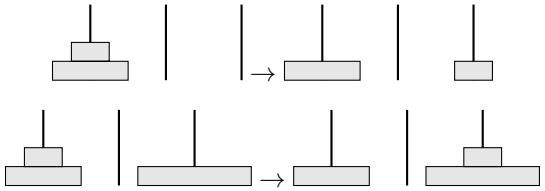
Q17 **C - 20min - 2pt** Écrire, en C, une fonction de signature `void crible(int p[], int N)` qui prend en entrée un tableau de $N+1$ valeurs entières (non initialisées) et les remplis de sorte que `p[i]` contienne 1 si i est premier et 0 sinon.
On appliquera la méthode précédente.
On essaiera de faire en sorte que la fonction n'utilise que des additions ou des multiplications par deux (ni multiplication, ni division, ni modulo). Si vous n'y arrivez pas, écrivez toute de même la fonction avec ces opérations (mais vous aurez moins de points).

3 Tours de Hanoi - 3pts

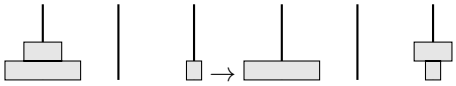
Le but de cet exercice est de résoudre le casse-tête suivant :



On a trois poteaux, numérotés 1, 2 et 3, sur lesquels on peut enfiler des disques.
Le but du jeu est de partir de la situation décrite sur la photo : tous les disques empilés sur le poteau 1 (à gauche) et d'aboutir aux disques empilés dans le même ordre mais sur le poteau 3 (à droite). Pour cela on ne peut faire qu'un seul coup : déplacer le disque au sommet d'un poteau vers un autre poteau. Par contre, il est interdit de placer un disque sur un disque plus petit.
Par exemple les coups suivants sont licites :



Mais celui-ci ne l'est pas :



On considère qu'au départ on dispose de n disques de tailles toutes différentes sur le poteau 1 et que les deux poteaux 2 et 3 sont vides. On cherche à déplacer tous les disques sur le poteau 3. Le poteau 2 joue un rôle crucial, il sert de espace de travail pour déplacer les disques.

Q18 **5min - 1/2pt** Pour quelle(s) valeur(s) de n peut-on se contenter de deux poteaux ?

Q19 **10min - 1/2pt** On suppose qu'on sait déplacer $n-1$ disques d'un poteau à un autre en utilisant un troisième poteau comme pivot, donner une stratégie pour résoudre le problème à n disques.

On suppose définie une fonction C `void deplace(int src, int dst)` qui déplace le disque du poteau numéroté `src` (un int) vers celui numéroté `dst`. Ainsi `deplace(1, 2)` déplace un disque du poteau 1 vers le poteau 2.
On indique également qu'il est possible d'écrire des fonctions récursives en C comme en Python en appelant juste la fonction comme dans :

```
unsigned int fact(unsigned int n)
{
    if (n == 0)
    {
        return 1;
    }
```

```
    else
    {
        return n * fact(n-1);
    }
}
```

Q20 **C - 10min - 1pt** Écrire une fonction `void hanoi(int n, int src, int dst, int inter)` qui résout Hanoi pour n disques en partant de src pour atteindre dst et utilisant inter comme pivot.

Q21 **10min - 1pt** Soit D_n le nombre de déplacements effectués pour déplacer n disques avec la fonction précédente. Déterminer une relation de récurrence satisfaite par D_n et en déduire l'expression de D_n en fonction de n.

4 Couvertures - 5pts

On considère une ligne de n cases carrés : ☐☐☐☐☐☐ que l'on souhaite recouvrir complètement avec des tuiles de différentes longueurs.
Par exemple, si on dispose de tuiles de longueur 1 ou 2, voici une couverture valide avec deux tuiles de longueur1 et deux de longueur 2. ▪☐▪☐

Q22 **OCaml - 10min - 1pt** Écrire une fonction OCaml `couvertures12 : int -> int` telle que `couvertures12 n` calcule le nombre de couvertures de n cases avec des tuiles de longueurs 1 ou 2.
Indication : on pourra raisonner sur la tuile la plus à gauche d'une telle couverture.

Q23 **2min - 1/4pt** En relisant la fonction précédente, reliez ce nombre de couvertures à une suite classique.

Q24 **5min - 1/2pt** Montrer qu'il est équivalent de couvrir n cases ainsi et de couvrir un échiquier de $2 \times n$ cases avec des dominos.
Rappel : un domino est un rectangle de deux cases de long, on peut donc le placer verticalement ou horizontalement pour couvrir deux cases de l'échiquier.

Q25 **10min - 3/4pt** Déterminer une condition sur n et m pour qu'un échiquier de $n \times m$ cases puisse être recouvert par des dominos.

Q26 **OCaml - 20min - 2pt** **(difficile)** Écrire une fonction OCaml `couvertures : int list -> int -> int` telle que `couverture l n` calcule le nombre de couvertures de n cases avec des tuiles de longueur données par la liste l.
Par exemple, la fonction précédente correspondra à `couvertures [1;2]`
On ne suppose rien sur l. Si un nombre est présent plusieurs fois, c'est comme si on considérait des tuiles de même longueur mais de couleurs différentes, ce qui augment le nombre de couvertures.

Q27 **5min - 1/2pt** Calculer le nombre de couvertures pour la liste `[1;1;1;1]` qui contient donc quatre fois une tuile de longueur 1 pour quatre couleurs différentes.



